MTA SZTAKI Közlemények 35/1986 pp.53-63

AN APPROACH TO AUTOMATED SYNTAX ANALYSIS

Lic. Miguel Fonfria ATAN
Lic. Eugenia G. Muniz LODOS
Lic. Luis A. Fajardo Alvarez de la CAMPA
Ing. Jorge L. de la Cantera RUIZ
Lic. R. Basilio Zubillaga BERAZAIN
Lic. Luis E. Fernandez LARA

Institute Central de Investigacion Digital
Habana, Cuba

Introduction.

In programming or writing compilers three problems must be solved: syntax analysis, lexicological analysis, and code generation or semantic management.

At the present level of development of the theory of languages and compilers, it is almost an established principle that from the three problems mentioned above only the third one requires a heavy treatment by the writers of compilers, i.e. the lexicon, and especially the syntax, should be a frame only for semantic management. The writer of compilers should find a syntactical method flexible enough to enable adequate semantic management.

In order to achieve the automation of syntax analyzers, a wide variety of syntax analyser generators have been developed from the description of a language grammar described in Backus's Normal Form, or in some other similar manner to provide the compiler writer with the syntax analyzer.

The method of syntax analysis can be divided into two large groups: the ascending ones which build the syntax tree of recognition from the chain been analyzed up to the generation's syntax auxiliary, and the descending ones which, starting from the generation's syntax auxiliary, work down to the text to be recognized.

Within the recognisers of the ascending type, techniques of precedence have been successful, and also that defined by Knuth in 1965, the LR technique, which has been widely accepted for theoretical and practical studies. From this, the techniques SLR and LALR, defined by De Remer in the late sixties, have been derived.

With particular reference to LALR(1) techniques, a syntax analyzer generator was implemented in 1971 starting from LALR(1) grammars, that is, grammars to which the LALR(1) technique is applicable, thus demonstrating the feasibility of generating analyzers of this type.

In 1972, a Syntax Analyzer Generator (SAG) similar to the former was built at the Centro de Investigacion Digital, from which tables were generated which enabled syntax analysis of ALGOL 60 and COBOL compilers for the CID 201-B minicomputer, the effectiveness of this method of analysis having been demonstrated in both.

The method of analysis LALR(1) in the form of a program guided by tables has the required flexibility as mentioned above. It possesses qualities which render it effective as a base for a SAG, among which are the following:

- Determinism in the analysis.
- Easy interaction with lexicologic and semantic management.
- Instantaneous detection of errors.
- Generality, in the sense that the class of LALR(1) grammars is wide.
- Easy alteration of language syntax.
- Easy recovery of syntax errors.
- Valuable parameters with respect to memory required and speed of analysis.

The method of analysis with LALR(1) grammars should be regarded as the programming of a deterministic pushdown automata that performs the syntax analysis by making adequated changes in its states.

The types of states present in this automata are three: applications, read, and look-ahead states. For each rule or production in the grammar there is a state of application, these states being the suitable ones to interact with the semantic management of the compilation process. The action that takes place in these may be resumed, from a properly syntactical viewpoint, into two tasks:

- Reducing or increasing the analyzer stack according to the number of symbols present on the right portion of the rule which produced it.
- Comparing the top of the stack with the first component of a set of associated pairs and comming up to the state indicated by the second component of the corresponding pair.

The reading states' function is to read the chain of which analysis is desired as to whether or not it belongs to the

language. The syntactical action associated with them is one of reading the symbol and comparing it with one or more that can occur in the state, comming up to the corresponding destination of the matched symbol. If the symbol being read is not among those legal in that state, a syntax error is detected.

The look-ahead states have their origin in the automata's need to look at a symbol in the chain (that is, the head) in order to determine which one is to be the next state that will enable proper continuation of the analysis. The associated syntactical action is to ascertain in what branch the looked symbol is located and come up to the corresponding destination. If the symbol is not found in any of the branches then a syntax error is detected.

These two states are the ones which interact with the lexicologic analyser in the compilation process. In conceiving the look-ahead states, it should always be remembered that these states do not read but only look at the symbol.

The method of analysis operates with a stack into which the reading states are pushed as they are consulted, the necessary history being maintained in the stack that enables continuation of syntax analysis.

SAG applications.

Our experience with the use of the SAG of CID 201-B began with the COBOL compiler written for the CID 201-B, with which the grammar corresponding to PROCEDURE DIVISION was processed. At that time the tables resulting from SAG was a listing that had to be manually loaded and optimized. Notwithstanding this

difficulty, the use of this technology increased the efficiency of the System and the speed of its setting. We will not go further into this application because the current version of SAG is an improvement by which, besides the listing with all the states, terminals, non terminals, etc., a paper tape is obtained containing the tables already coded and optimized.

In the case of the COBOL compiler for the CID 300/10 we decided to use this method for the syntax analysis of the whole language. Starting from the syntax of the COBOL sentence, the grammar for the language was designed. However, due to the structure of the compiler, which owing to memory capacity problems is divided into overlay regions, the language's grammar was divided into three parts: one for compiling IDENTIFICATION DIVISION and ENVIRONMENT DIVISION, one for compiling DATA DIVISION, and another one for PROCEDURE DIVISION. With each of these parts the SAG was used to obtain the tables and thus the corresponding syntax analyzer.

The procedure followed is simple: the compiler has an initial state to detected the symbol IDENTIFICATION, then it delivers control to the syntax analyzer of the first two divisions, which finishes upon detecting the terminal symbol DATA. Next the syntax analyzer of DATA DIVISION is loaded and given control, which ends when symbol PROCEDURE is detected, thus loading and delivering control to the syntax analyzer of this division, which finishes when end of compilation is detected.

The grammar of PROCEDURE DIVISION has a special characteristic in so far as it was necessary to divide the analyzer into two parts owing to memory capacity problems with CID 201-B when this

grammar was processed (it has the highest number of rules and is the more complex due to recursiveness). So there exists two grammars named "PROCEDURE P" and "PROCEDURE S", with which the syntax analysis of COBOL instructions is performed.

There were grouped in the "PROCEDURE P" grammar, the conditional sentences of COBOL and all grammatic rules presenting recursiveness while in the "PROCEDURE S" grammar the imperative sentences of the language were placed.

The main problem with this approach was the way in which the change from one table to another was to be carried out (in the same memory area) so that it should be transparent to the compiler. The following is a description of the solution adopted.

Syntax analyzer "PROCEDURE P" was the first to receive control, as has been explained, upon detection of the word PROCEDURE. For this analyzer, imperative sentences of the language such as ACCEPT, DISPLAY, etc., are terminals. This enables one state alone of "multiple reading" generated by the analyzer to detect the type of instruction being managed. If it was a conditional instruction, its analysis can be done within the analyzer "PROCEDURE P".

In case of an imperative instruction, when the word identifying the instruction is detected (it appears as a terminal), control is given to a special procedure (it appears in the compiler as one more semantic subroutine) that substitutes in the compiler the parameters of the tables generated by "PROCEDURE P" grammar by those of "PROCEDURE S", adequately placing certain indicators within the compiler, and continuing the processing by analyser

"PROCEDURE S". When the end of the instruction is detected. control is given to another special procedure (it appears in the compiler as a semantic subroutine) that carries out the process in reverse order, and processing then continues by analyser "PROCEDURE P".

It is to be noted that the side effect of this solution is a reduction in the speed of compilation, since the grammar tables have to be exchanged when an imperative instruction appears in the source program.

In this manner we were able to apply the SAG in the COBOL compiler for CID 300/10 and solve the problems.

With the experience obtained in applying the SAG extensive. complex grammars, we tackled the design of the compiler of the dBASE-300 laguage, having in mind to process its grammar in automated form. A grammar was designed that included all sentences in the language. This first grammar, however, could not entirely be processed by SAG. At this point we wish to comment the deficiencies and shortcomings of SAG so that the final decision adopted be understood.

The Syntax Analyser Generator (SAG) written for the minicomputer CID 201-B with 32 K words of central memory (like PDP-8), behaves like an independent program, that is, it operates with no Operating System. Due to memory restrictions, it was not possible to go deep into the diagnosis of errors; both the syntax errors of grammar (due to ambiguity of common errors in punching) and the error of exceeded capacity causes the program to stop execution without issuing any specific error message, so the user must go into an analysis of its whole grammar and find with

a pragmatic approach of "trial and error" a solution of the problem. This is a slow, tedious process in which the grammar must be rewritten again and again, and executed by SAG.

This process took, in general, more time than foreseen and we had to make a decision to obtain our syntax analyzer through the automated method (produced by SAG) and the ad hoc method.

To achieve this end we gradually reduced the initial grammar and finally obtained a grammar that can successfully be executed by SAG, the generated tables are loaded and the rest of the sentences are processed by the ad hoc method. To accomplish this it was necessary to alter both the scanner and the syntax analyzer so that it contemplated the work with the two methods.

Every command of the dBASE-300 language is processed by SAG except those that are simple (formed by terminals only) and the arithmetic expressions which upon the syntax analyzer detecting terminals with which an arithmetic expression can be started gives control to a module called arithmetic scanner that processes the expression by ad hoc method. When detecting a symbol not belonging to the expression the arithmetic scanner returns control to the syntax analyzer delivering the read symbol as not read so as not to affect the syntactic analysis.

Conclusions.

The use of a Syntax Analyser Generator allows to increase the performance of the programming techniques in any process that requires a complex syntactic analysis.

Using SAG for CID 201-B it was possible to verify the theoretical advantages that have been described in the introduction of this paper, although due to peculiarities of the implementation these advantages are reduced. In any case, it has been shown that its application may with satisfactory results be accepted as a "partial" solution for the syntax analysis of complex grammars.

Bibliography.

1.- Aho, A.V., J. Ullman The theory of Parsing, Translation and Compiling. Volume 1: Parsing. Prentice Hall, 1972.

2.- Anderson, T., J. Eve, J.J Horning. Efficient LR(1) parsers. University of Toronto, Computer Systems Group, 1972.

3.- De Remer, F.L. Simple LR(k) Grammars. Comm. ACM 14, 1971.

4.- Fonfria, M., E. Muniz, Informe tecnico del Compilador de COBOL para la CID 201-B.

5.- Fonfria, M. y otros, Cinco anos de Aseguramiento de Programas Basico de Gestion para el Sistema CID-1310, ICID, 1986.

6.- Fonfria, M. y otros, Sistema de Gestion de Bases de Datos dBASE-300. ICID, 1986.

7.- Gries, D. Compiler Construction for Digital Computers, New York. John Wley, 1971.

8.- Jares, L.R. A Syntax Directed Error Recovery Method. University of Toronto, Computer Sysytems Group, 1972.

9.- Lalonde, W.R. An Efficient LARL Parser Generator, University of Toronto, Computer Systems Group, 1971.

10.- Mc Keeman, W.M., J.J Horning, D.B. Wortma. A Compiler Generator. Englewood Cliffs, N.J., 1970.

11.- Muniz, E., V. Llopis, B. Zubillaga. Informe Tecnico del Generador de Analizadores Sintactico LALR(1), CID, 1976.

12.- Zubillaga, B. Generador de tablas optimizadas LALR(1), CID, Universidad de la Habana 1976 Tesis de Especialista.

13.- Zubillaga, B. Representacion optimizada de una Familia de Conjuntos. Rev. CID Electronica y Proceso de Datos en Cuba. No. 2 1982.

Az automatikus szintax-analizis egy tárgyalása

M.F. Atan, E.G. Lodos, L.A.F.A. de la Campa,
J.L. de la Cantera Ruiz, R.B.Z. Berazain,
L.E.F. Lara

Összefoglaló

1972-ben a Havannai Számitástechnikai Intézetben /Centro de Investigacion Digital/ épitettek egy szintax-analizis generátort /Syntax Analyzer Generator, SAG/, amely felhasználásával végezték el a CID 201-B mini-számitógép számára az ALGOL ill. COBOL forditóprogramjainak szintaktikus elemzését. A cikk a munka elméleti hátterét és a tapasztalatokat foglalja össze.

Подход к автоматическому синтакс-анализатору

М.Ф. Атан, Е.Г.М. Лодос, Л.А.Ф.А. де ла Кампа,
Й.Л. де ла Кантера Руиз, Р.Б. Беразаин,
Л.Е.Ф. Лара

Р е з ю м е

В 1972 г. был в Гаванском Вычислительном Институте /Centro de Investigacion Digital/ построен генератор синтакс-анализа /Syntax Analyzer Generator, SAG/, с помощью которого были анализированы компайлеры АЛГОЛ-а и КОБОЛ-а на машине CID 201-B. Статья описывает теоретические основы работы и опыты с работой.